九州大学学術情報リポジトリ
Kyushu University Institutional Repository

データパス分割に基づく空間的冗長性を利用した高信頼プロセッサ

松坂, 茂治
福岡大学大学院工学研究科電子情報工学専攻

井上, 弘士
福岡大学工学部電子情報工学科

https://hdl.handle.net/2324/6158

出版情報：第3回情報科学技術フォーラム(FIT2004), pp.273-274, 2004-09. 情報処理学会, 電子情報通信学会
バージョン：
権利関係：ここに掲載した著作物の利用に関する注意 本著作物の著作権は（社）情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。

データパス分割に基づく空間的冗長性を利用した高信頼プロセッサ
A Dependable Processor Architecture Exploiting Spatial Redundancy Realized by Datapath Partitioning

松坂 茂治†
Shigeharu Matsusaka

井上 弘士‡
Koji Inoue

1. はじめに

近年，コンピュータ・システムは様々な分野で利用されるようになった．特に，オンライン・トランザクション処理，人工衛星，医療，交通制御などでは，コンピュータ・システムの障害が重大な影響を及ぼすため，その高信頼化が必要不可欠となる．特に，コンピュータ・システムの核となるプロセッサの信頼性を向上することは極めて重要である．故障は，故障が長期間にわたって存在する永久故障（permanent fault）と，システムの稼動サイクルと比較して非常に短い時間にだけ存在する一時故障（temporary fault）に分けられる．永久故障の発生要因としては，半導体接合の破壊や回路短絡，断線などシステムの内部的要因が考えられる．一方，一時故障は，温度の変化や振動，α粒子や宇宙線などシステムの外部的要因が主な原因となる．

故障を検出する一般的な手法として，時間的または空間的冗長性の利用が挙げられる．時間的冗長性は，1 個のプロセッサで同じプログラムを複数回実行し，結果を比較することで故障を検出する．しかしながら，この場合には，実行時間の増加という問題が生じる．また，同一プロセッサを使用するため，永久故障を検出することはできない．一方，空間的冗長性は，複数個のプロセッサで同じプログラムを同時に実行し，結果を比較することで故障を検出する．この場合には，永久故障と一時故障の両方を検出できるが，ハードウェア・コストが大きくなる．

そこで本稿では，ハードウェア・コストの大幅な増大を招くことなく，空間的冗長性を実現するデータパス分割方式を提案する．また，演算時に必要となる最小ビット幅（以下，有効演算ビット幅と呼ぶ）を考慮し，実行時間オーバヘッドを削減する方式を示す．さらに，全ての演算に関する有効演算ビット幅は既知であると仮定し，データパス分割適用後の最小実行時間オーバヘッドを評価する．

2. データパス分割に基づく空間的冗長性の実現

本節では，データパス分割方式の詳細を説明する．なお，ここでは，データパス幅は 32 ビットと仮定する．

2.1 単純多重化方式

提案方式では，例えば既存の 32bit データパスを 2 つに分割し（各データパスのビット幅は 16bit になる），それぞれにおいて同じ演算を実行する．この様子を図 1[a]に示す．ここでは，3 つの演算命令が連続して実行される場合を想定しており，各命令の実行時間は 1 クロック・サイクル(CC)とする．冗長度が 1 の場合，32 ビット幅である従来のデータパスをそのまま使用するため，3 命令の実行時間は 3CC となる．これに対し，同一データパスにおいて空間的冗長度 2 を実現する場合，各命令の実行はそれぞれ 2 回の 16 ビット演算で実現される．同様に，空間的冗長度を 4 とする場合，各命令は 4 回の 8 ビット演算を要する．このようなデータパス分割方式を**単純多重化方式**と呼ぶ．単純多重化方式の欠点は実行時間がおよそ冗長度の倍数になる事である．例えば，冗長度が 2 の場合，3 命令の実行時間は約 2 倍(6CC)になる．これは，従来の時間的冗長性を活用した単一データパスでの複数回実行時と同程度のオーバヘッドである．しかしながら，時間的冗長性の活用とは異なり，単純多重化方式では永久故障の検出も可能となる．

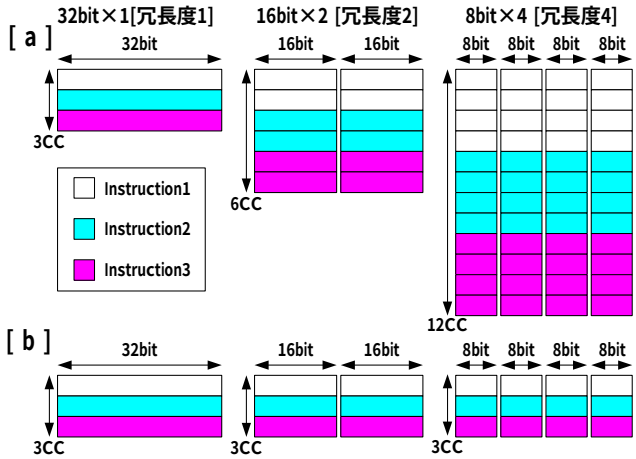

図 1:提案手法概念

2.2 圧縮多重化方式

第 2.1 節で述べたように，単純多重化方式の欠点は実行時間の増加である．そこで，この問題を解決する手段として**圧縮多重化方式**を提案する．通常，多くのプログラムにおいて，演算命令実行時に必要とされる有効演算ビット幅は，データパス幅より小さい場合が多い事が知られている．文献[3]では，このような特徴を利用して，プロセッサやメモリのビット幅を小さくする事で低消費電力化を実現している．これに対し，圧縮多重化方式では，プロセッサの信頼性向上を目的として有効演算ビットを活用する．圧縮多重化方式における実行の様子を図 1[b]に示す．空間的冗長度を 2 とする場合，各命令の有効演算ビット幅が 16 ビット以下であれば，図に示すように実行時間の増加を伴う事無く，分割された 2 つのデータパスで当該命令を実行できる．また，有効演算ビット幅が 8 ビット以下の場合，空間冗長度が 4 の場合でも実行時間オーバヘッドは発生しない．

† 福岡大学大学院工学研究科電子情報工学専攻
‡ 福岡大学工学部電子情報工学科

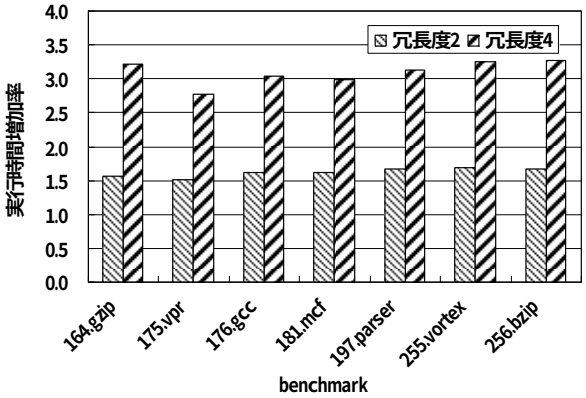

図 2:実行時間増加率

2.3 データパス分割方法

第 2.1 節ならびに第 2.2 節で示したデータパス分割方式の実現方法として，以下の 2 つが考えられる．一つは静的なデータパス分割である．この場合，プログラムコンパイル時にデータパス分割を見かけ上実現できるよう，オブジェクトコードを生成する．圧縮多重化方式を実現する場合，コンパイル時に有効演算ビット幅を解析しなければならない[2]．もう一つの実現方式は，動的なデータパス分割である．プログラム実行時，専用ハードウェアによって命令を分割実行する．圧縮多重化方式を実現する場合，有効演算ビット幅を動的に解析する回路が必要となる[1].

3. 性能評価

本節では，全ての有効演算ビット幅は既知であると仮定し，圧縮多重化方式における実行時間増加率を評価する．

3.1 実験環境

実行時間(*Execution Time*)は，次の式で与えられる．

*Execution Time = IC * CPI * CCT*

ここで，*IC* は実行命令数，*CPI* は命令当たりの平均所要クロックサイクル数，*CCT* :はクロックサイクル時間である．なお，本評価では CPI を 1 と仮定する．ここで，圧縮多重化方式を用いた冗長度 2 ならびに 4 を実現した場合，実行命令数は次の式から求められる．

$IC_{SR2} = IC_{press} + IC_{non\text{-}press} * 2$

$IC_{SR4} = IC_{press} + IC_{non\text{-}press} * 4$

- IC_{SR2}, IC_{SR4} :冗長度 2 または 4 の場合の実行命令数
- $IC_{non\text{-}press}$:「有効演算ビット幅>分割後のデータパス幅」であった実行命令数
- IC_{press} :「有効演算ビット幅≦分割後のデータパス幅」であった実行命令数

以上のような条件の下で，SimpleScalar ツール・セット(ver.3.0d)[4]を用いて，各命令実行における $IC_{non\text{-}pres}$ ならびに IC_{press} を測定した．なお，ベンチマーク・プログラムとしては SPEC2000[5]から 7 つの整数プログラムを用いて実行した．なお，入力には small インプットを使用している．

3.2 実験結果

図 2 に各ベンチマークにおける実行時間増加率を示す．結果は，冗長度が 1 である従来型プロセッサでの実行時間で正規化している．第 2 節で述べたように，単純多重化方式を用いると，実行時間は冗長度 2 の場合で 2 倍，冗長度 4 の場合で 4 倍とる．これに対し，圧縮多重化手法を用いると，冗長度が 2 および 4 の場合で，それぞれ、平均 1.62 倍および 3.09 倍であった．

表 1:圧縮可能命令率

	全命令中の内訳	圧縮可能命令率	
		冗長度2	冗長度4
分岐命令	18.23%	82.43%	75.61%
load/store命令	37.32%	0.00%	0.00%
ALU命令	41.59%	52.62%	36.32%
その他	2.86%	0.00%	0.00%

実行時間の増加をより詳細に解析するため，全命令実行の内訳(分岐命令・load/store 命令・ALU 命令・その他の命令)，ならびに，それらにおける圧縮可能命令の出現率を測定した．その結果を表 1 に示す（全ベンチマークの内訳）．例えば，実行された全命令の 41.59%は ALU 命令であり，その内の 52.62%が圧縮可能（つまり，有効演算ビット幅が 16 ビット以下）である．表 1 より，最も実行回数が多い ALU 命令に関しては約半分（冗長度が 2 の場合）程度の命令に関して圧縮可能である．しかしながら，全実行の約 37%を占める load/store 命令に関しては殆ど圧縮できていない．これは，load/store 命令を実行する際，ベース・アドレスが大きな値であったためである．よって，実行時間の増加を更に低減するには，コンパイラによるデータ再配置や，ベース・アドレス専用圧縮方式を確立する必要がある．

4．おわりに

本稿では，ハードウェア・コストの大幅な増大を招くことなく，空間的冗長性を実現するデータパス分割方式を提案した．また，最善ケースを想定し，実行時間増加率を評価した．その結果，空間的冗長度が 2 の場合で平均 1.62 倍，4 の場合で平均 3.09 倍の実行時間増加である事が分かった．今後，データパス分割をサポートしたプロセッサの設計，ならびに，第 2.3 節で説明したデータパス分割実行の実現方式を確立する予定である．

謝辞

多くの有用なアドバイスを頂いた福岡大学モシニャガ・ワシリー教授に感謝します．なお，本研究は一部，科学研究費補助金(課題番号：14GS0218, 14702064）による．

参考文献

[1] D. Brooks and M. Martonosi. “Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance.” *HPCA*, pp.13-22, 1999.

[2] H. Yamashita, H. Yasuura, F. N. Eko, and C. Yun, “Variable Size Analysis and Validation of Computation Quality”, *Proc. of Workshop on High-Level Design Validation and Test* , pp.95-100, Nov. 2000.

[3] Y. Cao and H. Yasuura. “A System-level Energy Minimization Approach Using Datapath Width Optimization,” *Int. Symp. on Low Power Electronics and Design*, pp.231-236, Aug. 2001.

[4] SimpleScalar Tool Sets, http://www.simplescalar.com/.

[5] SPEC (Standard Performance Evaluation Corporation), http://www.specbench.org/.